NASA-CR-203073

NASA/WVU Software IV & V Facility
Software Research Laboratory
Technical Report Series

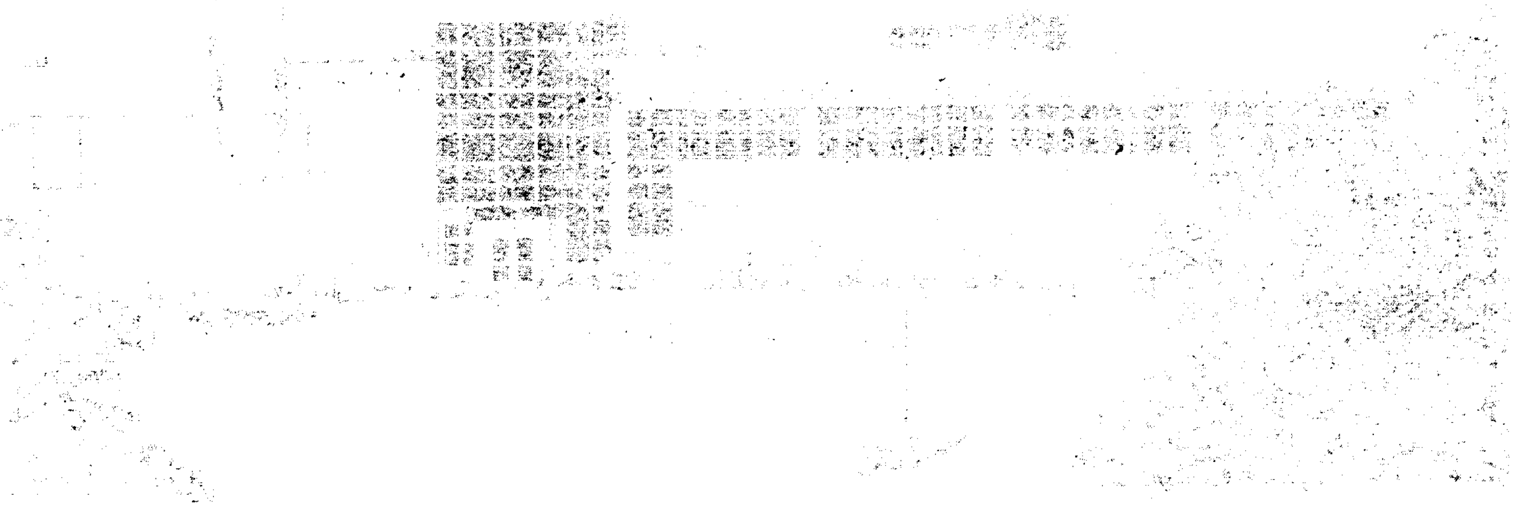
NASA-IVV-96-004
WVU-SRL-96-004
WVU-SCS-TR-96-13
CERC-TR-RN-96-008

IN-61-CR

008161

The Applicability of Proposed Object-Oriented Metrics to Developer Feedback in Time to Impact Development

by Ralph D. Neal



National Aeronautics and Space Administration

West Virginia University

According to the terms of Cooperative Agreement #NCCW-0040, the following approval is granted for distribution of this technical report outside the NASA/WVU Software Research Laboratory

[illegible]	[illegible]
George J. Sabolish Date	John R. Callahan Date
Manager, Software Engineering	WVU Principal Investigator

The Applicability of Proposed Object-Oriented Metrics to Developer Feedback in Time to Impact Development

Ralph D. (Butch) Neal
NASA Software IV&V Facility
West Virginia University

Abstract

This paper looks closely at each of the software metrics generated by the McCabe Object-Oriented Tool™ and its ability to convey timely information to developers. The metrics are examined for meaningfulness in terms of the scale assignable to the metric by the rules of measurement theory and the software dimension being measured. Recommendations are made as to the proper use of each metric and its ability to influence development at an early stage. The metrics of the McCabe Object-Oriented Tool™ set were selected because of the tool's use in a couple of NASA IV&V projects.

Ralph D. (Butch) Neal
Concurrent Engineering Research Center
NASA/WVU Software IV&V Facility
100 University Drive
Fairmont, WV 26554

Phone:	304 367-8355
Fax:	304 367-8211
e-mail:	rneal@cerc.wvu.edu
www:	http://research.ivv.nasa.gov/~rneal

The Applicability of Proposed Object Oriented Metrics to Developer Feedback in Time to Impact Development

Abstract

This paper looks closely at each of the software metrics generated by the McCabe Object-Oriented Tool™ and its ability to convey timely information to developers. The metrics are examined for meaningfulness in terms of the scale assignable to the metric by the rules of measurement theory and the software dimension being measured. Recommendations are made as to the proper use of each metric and its ability to influence development at an early stage.

1. Introduction

The proper function of independent verification and validation (IV&V) is the timely feedback into the development process of possible as well as actual problems. One mechanism for recognizing problems is measurement of leading indicators. Software measurement is too new to have empirically tested and verified measures that predict future problems. However, the discipline has started to design metrics that can be used in empirical experiments that will result in a valid suite of prediction measures.

The dangers of randomly (or carelessly) selecting predictor variables to include in statistical processes in an attempt to show causality are well known, e.g., things like, the Democrats always win the Presidency when the American League wins the World Series. However, the same sort of practice has been accepted in the software engineering community. Metrics are accepted as predictors with little or no theoretical validation. Fenton [1991] warns against accepting metrics as valid in the wide sense (valid as predictors) without validating them in the narrow sense (valid theoretical underpinnings). Following the lead of Fenton, we will refer to unvalidated measurements as metrics. A metric becomes a measure when it has been validated to actually measure some dimension of the software. Metrics which may be useful but cannot be validated as measures are called *red light* indicators.

The metrics of the McCabe Object-Oriented Tool™ set were selected because of the tool's use in a couple of NASA IV&V projects.

2. Background

2.1. Measurement Theory

Mathematical (and statistical) operations always can be performed on metrics. The question is, do the results make meaningful statements about the objects being measured? [Roberts, 1979]

When groups of objects are measured on the nominal scale: many statistics can not be used; the mode is the only meaningful measure of centrality. When groups of objects are measured on the ordinal scale: rank order statistics and non-parametric statistics can be used (assuming that the necessary probability distribution can be reasonably assumed to be present); the median is the most powerful meaningful measure of centrality. When groups of objects are measured on the interval scale: parametric statistics as well as all statistics that apply to ordinal scales can be used (it must be reasonable to accept that the necessary probability distribution is present); the arithmetic mean is the most powerful meaningful measure of centrality. When groups of objects are measured on the ratio scale: percentage calculations as well as all statistics that apply to interval scales can be used; the arithmetic mean is the most powerful meaningful measure of centrality.

2.2. Object-Oriented Paradigm

Authors have not been in agreement about the characteristics that identify the object-oriented approach. Henderson-Sellers [1991] listed information hiding, encapsulation, objects, classification, classes, abstraction, inheritance, polymorphism, dynamic binding, persistence, and composition as having been chosen by at least one author as a defining aspect of object-orientation. Rumbaugh, et al. [1991] added identity, Smith [1991] added single type and Sully [1993] added the unit building block to this list of defining aspects.

The old software metrics do not take these new concepts into consideration. Therefore, these characteristics necessitate the advent of new metrics to measure object-oriented software. We must find measures for modularity, cohesiveness, abstraction, polymorphism, data control, and inheritance to compliment the legacy measures for size, psychological complexity, and structural complexity.

2.3. Impact

The earlier a potential problem can be brought to the attention of the developer, the cheaper is the cost of fixing the problem [Boehm, 1981]. In order to assist the developer, measurements must be attainable from requirements or design documents.

3. The metrics

The McCabe Object Oriented Tool™ generates sixteen metrics. We'll look at each of these metrics and determine its usefulness in alerting developers to potential problems.

3.1. Average v(G) -- v(G) [McCabe, 1976]

The Average v(G) is the average cyclomatic complexity of the methods in a class. The v(G) of a method is the number of independent paths through the method. In structured methods, v(G) = the number of decision nodes plus one. Average v(G) is offered as a measure of structural complexity. V(G) only can be used as an ordinal scale [Zuse, 1990]. The most meaningful measure of centrality for a measure that defines an ordinal scale is the median, i.e., measures that define an ordinal scale cannot be used to calculate means (a measure on an ordinal scale cannot be summed). Average has many definitions. If the "average" in average v(G) is the median, then this measurement would be theoretically correct. However, v(G) is still of dubious value for small classes. If the "average" in average v(G) is the mean, this measurement is theoretically meaningless. In software development, it is often the legitimate outlier that we want to identify. Therefore, a theoretically better measure is Maximum v(G) of a class. Maximum v(G) allows development effort to be concentrated on the most complex method instead of spread out across the entire class. V(G) might prove to be of value in predicting maintenance effort required for individual methods but the Average v(G) is not informative for developer feedback.

3.2. Coupling Between Objects (CBO) [Chidamber and Kemerer, 1994]

CBO is a count of the number of distinct noninheritance related classes on which the measured class depends. CBO is presented by the authors as a measure of reusability. As reusability increases, CBO decreases. CBO can only be used as an ordinal scale [Neal, 1996]. According to Chidamber and Kemerer, 1) excessive coupling among object classes can hinder reuse through the deterioration of modular design, 2) the greater the degree of coupling the more sensitivity to changes in other parts of the program. CBO may be an indicator of *inter-object complexity*. While averages cannot be taken, individual measures may be used to trigger further study of an individual class' modularity. There are no baseline studies to indicate where the cut-off point might be between acceptable coupling and unacceptable coupling. The best use of the ordinal measure might be realized by comparing the classes of a system to each other. At least as an initial cut, this metric could be fed back to developers as an indicator that individual classes should be investigated as to the degree of independence.

3.3. Depth of Inheritance Tree (DIT) [Chidamber and Kemerer, 1994]

Depth is the level of a class within its inheritance tree. DIT is presented as a measure of complexity. In the light of measurement theory, DIT does not meet the requirements of even the ordinal scale [Neal, 1996]. DIT is a surrogate for the number of ancestor classes that could affect a class. However, DITs within a given project often cluster around one level and thus fail to discriminate from one class to another. However, the theoretical failings of this metric do not mean that it cannot be a useful indicator. Precisely because the values of DIT tend to cluster around very small integer numbers, outliers show a cause for concern. There are no baseline studies to indicate the cut-off point for depth of inheritance. The best use of DIT is to compare each class' value to the values of all other classes. Any class which falls well above the mode value should be investigated for over-design. This metric could be fed back to developers as a *red flag* that individual classes should be investigated as to the degree of design complexity.

3.4. Fan In (FI) [Henry and Kafura, 1981]

FI is the count of the parents of a class. McCabe offers FI as a measure of the complexity of the class brought about by multiple inheritance. However, FI cannot be accepted as an ordinal scale of complexity [Neal, 1996]. Again, the theoretical failings of this metric do not mean that it cannot be a useful indicator. A high FI may be an indicator of possible design flaws. McCabe recommends a threshold of two for this metric. At least until empirical evidence is available to indicate otherwise, this metric could be fed back to developers as an indicator that individual classes should be investigated as to class design.

3.5. Lack of Cohesion of Methods (LCOM) [Chidamber and Kemerer, 1994]

The definition given by McCabe is not the same as the definition given by Chidamber and Kemerer. The original has an artificial floor (zero) which keeps it from being very useful. McCabe describes the metric as: 100 minus the class mean of the percentage of methods using each variable. One need not calculate the percentages for each variable but can count all variables across all classes, multiply by 100, and divide by ((the number of individual variables) times (the number of classes)) and finally subtract from 100.

LCOM is presented by Chidamber and Kemerer as a measure of cohesiveness of methods within a class. Is this metric a measure or an indicator? Its unclear at this point. Proportions are ratio scales [Roberts, 1976]. Therefore, means of proportions should be ratio scales. Thus LCOM as defined by McCabe would fit the definition of a ratio scale and be considered a measure of lack of cohesion. I believe that this metric may be useful in the long run. Empirical tests are needed to know for sure. In the short term, it may be difficult to decide what value signifies the break between cohesive classes and uncohesive classes. It may also be difficult to obtain the counts in time to impact the development effort.

3.6. Max ev(G) -- ev(G) [McCabe, 1976]

Max ev(G) is the v(G) of a flowgraph measured after subroutines have been reduced to single nodes. The ev(G) of a structured program is one. In less structured programs, ev(G) takes on larger numbers. In object-oriented programs, ev(G) is calculated for each method. McCabe defines max ev(G) as the sum of the maximum essential complexity of each class in a system divided by the number of classes. Max ev(G) is a measure of structural complexity. According to the rules of measurement theory, ev(G) is an ordinal scale and therefore not additive [Zuse, 1990].

As an ordinal scale, the maximum ev(G) for individual classes is meaningful. The maximum ev(G) of individual classes would be a very informative measure. The developer needs to spot outliers. Averaging all classes together hides these outliers. So, average Max ev(G) is not useful to the developer and should not be used for feedback to development teams but maximum ev(G) for each class is useful and could be used as feedback to development teams.

3.7. Max v(G) -- v(G) [McCabe, 1976]

The v(G) of a program is the number of independent paths through the program. In structured programs, v(G) equals the number of decision nodes plus one. In object-oriented programs, v(G) is calculated for each method. McCabe defines max v(G) as the sum of the maximum cyclomatic complexity of each class in a system divided by the number of classes. Max v(G) is presented as a measure of structural complexity. According to the rules of measurement theory, v(G) is an ordinal scale and therefore is not additive [Zuse, 1990].

As an ordinal scale, the maximum v(G) for individual classes is meaningful. The maximum v(G) of individual classes would be a very informative measure since the developer needs to spot outliers. Averaging all classes together hides these outliers. So, average Max v(G) is not useful to the developer and should not be used for feedback to development teams but maximum v(G) for each class is useful and could be used as feedback to development teams. See also metric #1.

3.8. Number of Children (NOC) [Chidamber and Kemerer, 1991]

NOC is the count of the immediate subclasses of a class. NOC is a surrogate for the number of classes that might inherit methods from a parent. According to Chidamber and Kemerer, 1) the greater the number of children, the greater the inheritance and 2) the more children a parent class has, the greater the potential for improper abstraction of the parent class. NOC is presented by Chidamber and Kemerer as a measure of complexity.

Based on measurement theory validation, NOC may be used as an ordinal scale of psychological complexity (understandability) [Neal, 1996], i.e., the understandability of a class may well be related to the number of immediate subclasses. Furthermore, there may be a maximum NOC above which a class should be reviewed for the misuse of subclassing. This metric could be fed back to developers as a *red flag* that parent classes and their children classes should be investigated as to the degree of abstraction. Empirical evidence is needed to determine the long run usefulness of this measure.

3.9. Percent Overloaded Calls (POC) [unattributed]

POC is the percentage of calls that are made to overloaded modules. According to McCabe, this is a measure of the generality of the system, i.e., the higher this metric the more reusable the objects of the system. There is no empirical evidence to substantiate McCabe's claim. However, percentages are ratio scales [Roberts, 1976]. Ratio scales are the most powerful scales. Thus, POC would seem to be a good metric for the design team. This metric should be reevaluated when we have empirical data to study.

3.10. Percent Public/Protected (PP/P) [unattributed]

PP/P is the percentage of PUBLIC and PROTECTED data in a class that is directly accessible to objects or functions of the class. According to McCabe, this is a measure of the lack of encapsulation of the data, i.e., the higher this metric the less control each class has of the data in the class. There is no empirical evidence to substantiate McCabe's claim. However, percentages are ratio scales [Roberts, 1976]. Ratio scales are the most powerful scales. Thus, PP/P would seem to be a good metric for the design team. This metric should be reevaluated when we have empirical data to study.

3.11. Access to Public Data (APD) [unattributed]

APD is the count of the number of times that a class's PUBLIC and PROTECTED data is accessed by other classes. This metric, along with the previous metric, would seem to be a measure of data control. Studies of other counts have shown inter-class measures to be ordinal scales [Neal, 1996].

A high APD indicates that a larger segment of the system may be affected when changes are made than would be necessary if the data were PRIVATE. This metric should be fed back to developers so that the classes can be ranked by the degree of data control. This allows the classes to be investigated in descending order of rank. By taking the classes in descending order of rank, the worst classes receive the most and fastest attention. Metrics P/PP and APD seem to work together to analyze data design.

3.12. Quality [unattributed]

Quality (a misnomer if I ever saw one) is defined by McCabe as the number of classes dependent on descendants. Properly designed classes should not access their descendants, e.g., their children. Properly designed classes access only their ancestors, e.g., their parent(s). Therefore, this metric is an attempt to measure design. However, this metric is more correctly defined as a switch rather than a measure. Since properly designed classes never access their children, the switch of the metric from zero to one causes the developer to be alerted to possible design problems. This metric could be fed back to developers as an indicator that classes are accessing their descendant (children) classes and therefore should be investigated for design faults.

3.13. Response for a Class (RFC) [Chidamber and Kemerer, 1991]

RFC is a count of inherited methods plus a count of the unique outside methods invoked by the measured class, i.e., if an outside method is invoked more than once it is none-the-less counted only once. This metric has been proposed as a surrogate for the potential communication between the class and other classes and as such a measure of complexity. If understandability (psychological complexity) is the complexity being measured, and one accepts that once a method is understood, overall understandability does not vary with the number of times the method is invoked, then RFC could be accepted as an ordinal scale [Neal, 1996]. However, if structural complexity is being measured, we cannot accept that complexity remains unchanged when methods that invoke methods from other classes are added to the measured class. This is true even if the invoked methods are already being invoked by another method in the measured class. What this all means is, RFC is not a good measure of structural complexity but may be a good measure of understandability. RFC cannot be summed across classes nor can averages of RFC be calculated other than average in the sense of taking the median [Neal, 1996]. However, RFC does allow the developer to rank classes in order of complexity in the sense of understandability. At least as an initial cut, this metric could be fed back to developers to indicate what classes should be investigated first as to possible problems of psychological complexity. By taking the classes in order of complexity, the worst classes receive the most and fastest attention.

3.14. Number of Roots (NOR) [unattributed]

NOR is a count of the distinct class hierarchies utilized by a program. According to McCabe, this is a measure of the lack of inheritance, i.e., a higher NOR indicates that advantage is not being taken of similarities between classes. There is no empirical evidence to substantiate McCabe's claim. Is a program made up of fifty classes with a NOR of ten more guilty of ignoring inheritance than a program with ten classes and a NOR of five? We really don't know. There is no indication in NOR of the relative size of the program. This ambiguity keeps NOR from being a measure of the lack of inheritance. The theoretical failings of this metric do not mean that it cannot be a useful indicator. A high NOR may be an indicator of possible design flaws. McCabe recommends that NOR be used with DIT to evaluate a program. If DIT is low (the hierarchy chart is shallow) and NOR is high (the hierarchy chart is wide) it may indicate that similarities between classes are not being exploited. This metric could be fed back to developers as an indicator that programs with relatively high NOR should be investigated as to class design but NOR should be reevaluated when we have empirical data to study.

3.15. Sum v(G) [McCabe, 1976]

Sum v(G) is the sum of the cyclomatic complexities of the methods within a class. V(G) can be used only as an ordinal scale [Zuse, 1990]. Therefore, according to the rules of measurement theory, v(G) is not additive [Zuse, 1990]. See also metrics #1 and #7.

3.16. Weighted Methods per Class (WMC) [Chidamber and Kemerer, 1994]

WMC is the sum of weighted methods in a class, i.e., each method within the class is weighted by some sort of complexity metric and this weight is summed to arrive at WMC. WMC is presented by the authors as a measure of complexity. The problem with WMC, as proposed by Chidamber and Kemerer, is that the complexity metric is not defined other than to say that it should have the properties of the interval scale. The hard part might be finding such metrics. Zuse [1990] validated 98 complexity metrics for type of scale but chose not to validate any of them for the interval scale because of the difficulty of proof. Chidamber and Kemerer avoid the issue by showing in their example that complexity of each method can be assigned unity and WMC then becomes a count of the methods within the class. Churcher and Sheppard [1995] could find no better metric to use than to take Chidamber and Kemerer's advice and assign unity to each method. McCabe seems to have done the same. Defined thus, WMC is an ordinal scale. This metric could be fed back to developers as a *red flag* that classes should be investigated as to the degree of structural complexity or as an indicator that further decomposition is needed.

4. Summary

The metrics of The McCabe Object-Oriented Metrics Tool™ are skewed toward the measurement of complexity. The metrics are also skewed toward the legacy measurements of preobject-oriented systems. Broken down by assignable scale (and therefore by the meaningful data that they convey) the sixteen metrics look like this:
Four of the sixteen metrics cannot be assigned a scale and therefore are not useful for any type of measurement.

One metric can be used as a switch to alert developers to possible design problems.

Four metrics are useful indicators of possible problems even though they cannot be assigned scale. These metrics are useful for looking for outliers.

Two metrics are ordinal scales but are probably most useful as indicators. Although non-parametric statistics could be applied to these metrics, they are probably best used for rank order statistics.

Two more metrics are ordinal scales and are probably most useful as ordinal scale, i.e., they can be used for non-parametric as well as rank order statistics.

Three metrics are ratio scales and can be used for parametric statistical analysis.

Broken down by the software dimension (modularity, data control, inheritance, cohesiveness, and data abstraction for object-oriented programs and size, psychological complexity, and structural complexity for legacy programs) the metrics look like this:

One ratio scale each for cohesiveness, data abstraction, and data control.

One ordinal scale each for data control and psychological complexity.

Two ordinal scales that are best used as indicators for psychological complexity.

Three indicators for structural complexity.

One indicator and one switch for modularity.

It is obvious that many of the dimensions of software are not being measured by this tool while other dimensions are being over-measured. Measuring a dimension by more than one method is wasteful of time and resources. Metrics should be selected to cover as many of the dimensions as the developer feels is important to the system being developed. Some dimensions may not need to be measured. Keeping the measurement to a minimum while covering all of the important dimensions is the most cost effective approach.

Good object-oriented metrics have yet to be devised, tested, and proven. But, you have to start somewhere.

Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

Chidamber, Shyam R., and Chris F. Kemerer, *A Metric Suite for Object Oriented Design*, IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994.

Churcher, Neville I., and Martin J. Shepperd, *Towards a Conceptual Framework for Object Oriented Software Metrics*, ACM SIGSOFT Software Engineering Notes, Vol. 20, No. 2, April 1995a.

Fenton, Norman, Software Metrics: A Rigorous Approach, Chapman & Hall, London, UK, 1991.

Henderson-Sellers, B., A Book of Object-Oriented Knowledge, Prentice Hall, NY, 1992.

Henry, S. and D. Kafura, *Software Metrics Based on Information Flow*, IEEE Transactions on Software Engineering, Vol. 7, No. 5, 1981.

McCabe, T. J., *A Complexity Measure*, IEEE Transactions on Software Engineering, Vol. 5, 1976.

Neal, Ralph D., The Validation by Measurement Theory of Proposed Object-Oriented Metrics, Dissertation, Virginia Commonwealth University, Richmond, Va., 1996.

Roberts, Fred S., Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences, Addison-Wesley Publishing Company, Reading Massachusetts, 1979.

Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, Object-Oriented Modeling and Design, Prentice Hall, Englewood Cliffs, NJ, 1991.

Smith, David N., Concepts of Object-Oriented Programming, McGraw-Hill, NY, 1991.

Sully, Phil, Modeling the World with Objects, Prentice Hall, NY, 1993.

Zuse, Horst, Software Complexity: Measures and Methods, Walter de Gruyter, Berlin, 1990.